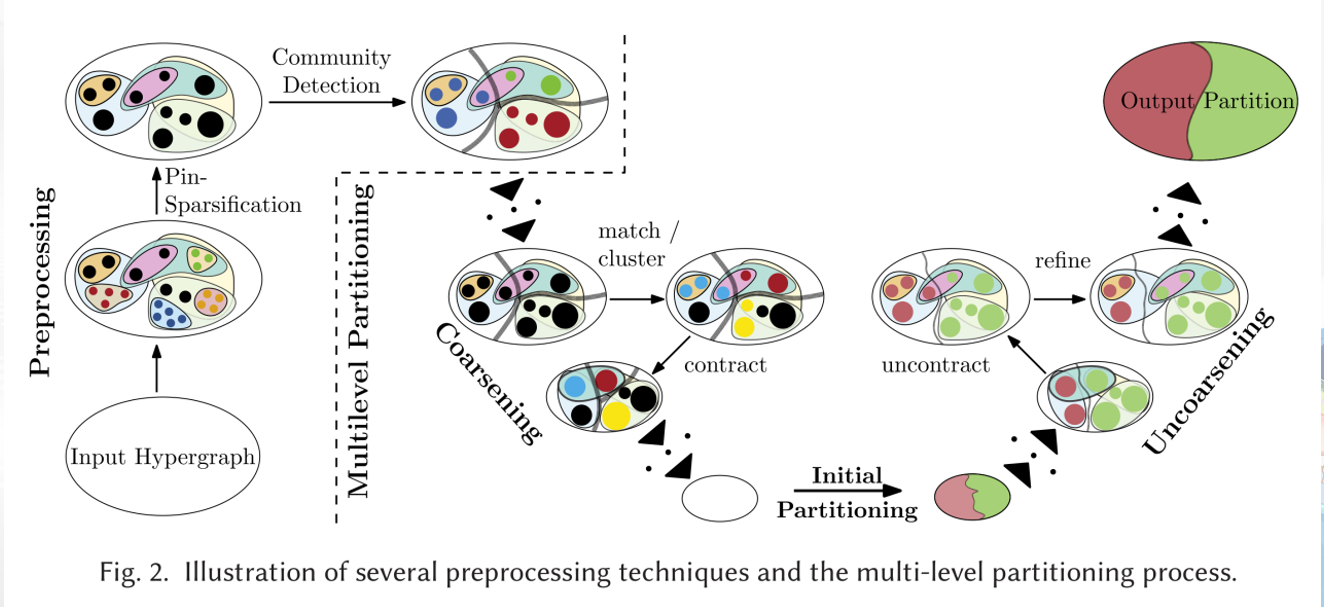
题目

摘要

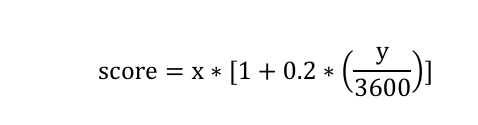
介绍

随着集成电路设计规模越来越大，硬件仿真平台难以用单个FPGA 仿真高达数十亿门规模的设计。因此，我们需要应用多FPGA系统处理此类工程。在这样的情景下，分割算法的质量和速度成为了新的挑战，一个好的分割算法可以在可控的时间内给出一个对时序优化有利的且易于FPGA内部布局布线的分割结果。如果不能保证合理的分割结果，那么电路的运行频率会受到极大的影响。 与此同时，对于部分特定电路逻辑的复制也可用来辅助优化分割结果。题目的核心在于将电路建模为超图，在满足约束条件的情况下进行超图分割，运用课程中Partitioning的知识，在时间和hop路径之间进行权衡来取得更高的分数。我们基于题目的要求提出了一个原创的基于多评分函数的贪心分割算法。流程图如下，有预处理、粗化、初始分割、解粗化和调整等步骤，在后面的章节中会一一介绍；



题目要求

题目给出了如下评分函数（x为hop路径，y为时间，单位是秒）



从函数中可以看出，我们的优化目标就是hop路径与时间。

同时题目也提出了如下限制：

hop路径：不得超过给出的最大hop路径

时间：不得超过一个小时

内存：不得超过32GB

线程：最多使用4个线程运行

资源：使用的FPGA资源数不得超过给定上限

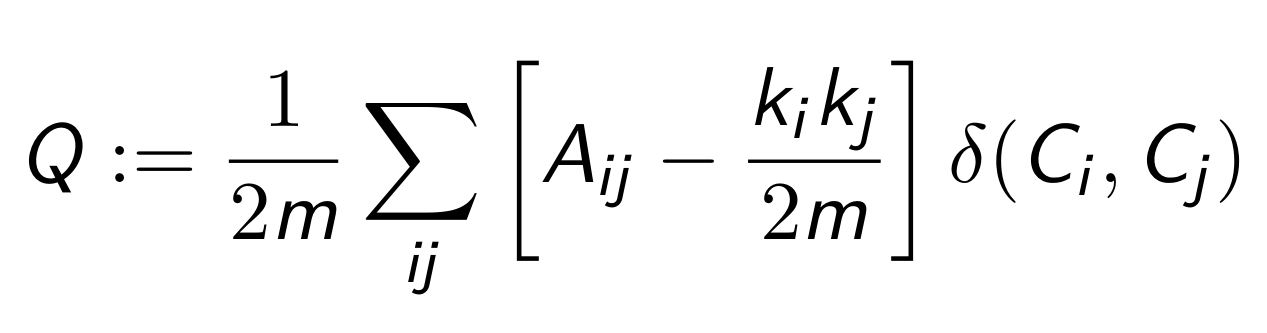
算法介绍

1. 预处理

在预处理阶段我们采用两种方法：

1.对节点进行多轮的局域敏感哈希，将具有相同哈希值的节点进行缩点；

2.使用Louvain算法将节点分为多个社区，此后的粗化过程只在社区内进行。



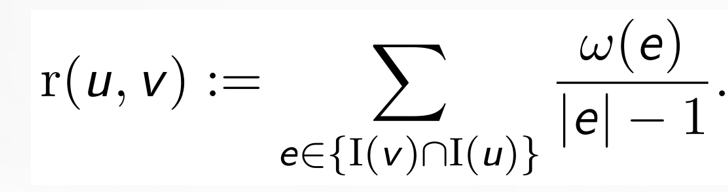
在经过试验后，我们放弃了预处理的步骤，有如下原因：

1.我们采用的哈希方法并不能大幅减小点的数目，如果放宽缩点的要求，又会超出很多限制对后续的步骤造成麻烦；

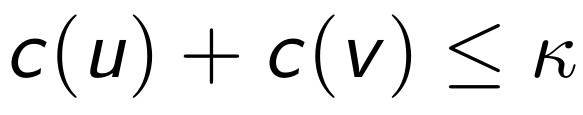
2.用Louvain直接得到的社区大小不均，只在社区内缩点同样无法达到缩小问题规模的目的，并且容易掉入局部最优，反而不如对全局节点一起进行粗化。

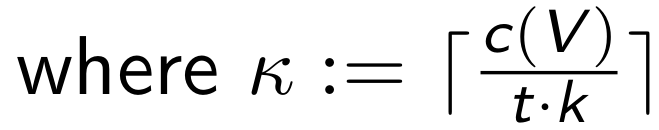
1. 粗化

我们使用一个带权重惩罚的评分函数来计算节点对之间的评分，惩罚机制使得网络中连接更多节点的边在计算评分时产生较小的贡献，这种机制确保超图在粗化过程中，节点合并不会过于依赖于高权重的边，从而使得合并过程更平衡，避免对某些网络的过度依赖。



我们将评分较高的节点对进行合并，直到满足我们的停止条件





这里的c包括节点数量和FPGA的各种资源，t为我们设置的约束参数，用来实现资源调节的松紧度，这样能够防止粗化过度，在初始分割中得到一个相对不错的解。

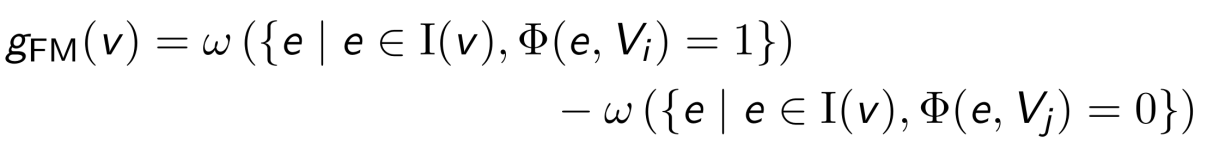
1. 初始分割

我们使用了三种分割方法：

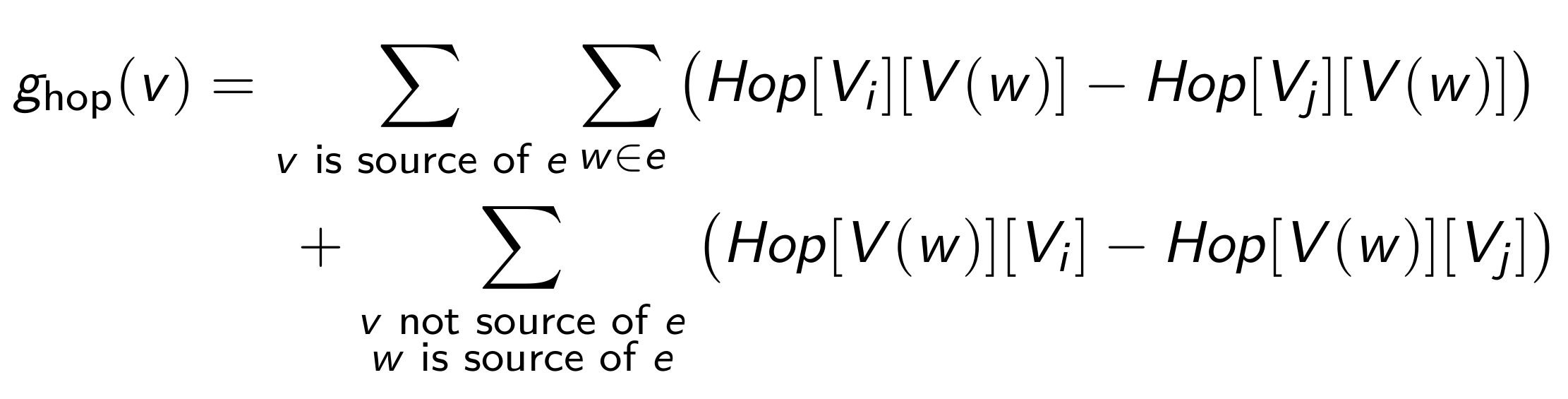
1. BFS：随机选取k个节点分别分配给k个FPGA，同时开始bfs，把到达的节点分配给相同的FPGA；
2. 标签传播：随机选取k个节点分别分配给k个FPGA，同时将它们的τ个邻居分配给相同的FPGA，然后进行多轮随机访问，将每个节点分配给它的邻居们的FPGA中能带来最大的FM/Hop收益的FPGA；
3. 贪心分割：对于每个分块，算法维护一个优先队列（PQ），存储根据评分函数计算的邻接节点。算法会迭代选择评分最高的节点，将其移动到对应分块中，并更新邻接节点的评分。

我们在贪心算法中，测试了多种评分函数，最后确定交替使用如下两种：

1. FM Gain：以最小化割边和为优化目标



1. Hop Gain：直接以最小化Hop和为优化目标



交替使用这两种评分函数可以兼顾分割的均衡性和Hop和的优化，有助于跳出局部最优得到更好的解。

在实际流程中，我们会根据电路规模进行一定次数的BFS和标签传播，保留结果最好的初始分割，然后在其基础上进行一定次数的贪心分割，得到最终的初始分割结果。

在初始分割后，查找全图中超出了max hop的节点，激活它们及其邻居，然后对他们继续进行几轮贪心分割，从而得到一个合法的分割结果，

3. 解粗化

在每次解除一个缩点后，我们激活两个节点，并计算它们移动到相邻 FPGA 的增益。然后将增益最大的节点移动到目标 FPGA，并激活其邻居节点，直到经过 log\_2n步后无法获得正增益，最后恢复到最大增益状态。

在节点较少时此方法表现良好，但随着电路规模的增大，计算量会急剧增加，尤其在稠密超图中，即使初始激活的节点很少，最终也会涉及大量节点，导致指数级的计算增长。

为了解决这个问题，我们通过缩减计算量来换取时间效率。虽然这会在某些情况下牺牲一定的解质量，但在大规模电路中，这种trade off对结果的影响较小。具体来说，我们采用了以下几种策略，并根据问题规模灵活组合使用：

1. 每次解除多个缩点，减少整体计算的次数；
2. 只激活每个邻居节点中的20-50%，而不是全部激活；
3. 进一步仅激活初始被激活的节点的邻居节点；
4. 使用优先队列存储增益，而不更新 Hop Gain；
5. 一旦出现负增益，立即结束。
6. 逻辑复制与多线程

首先我们搜索所有可能的逻辑复制方案，然后按照以下顺序进行复制：

1按照Hop增益从大到小顺序；

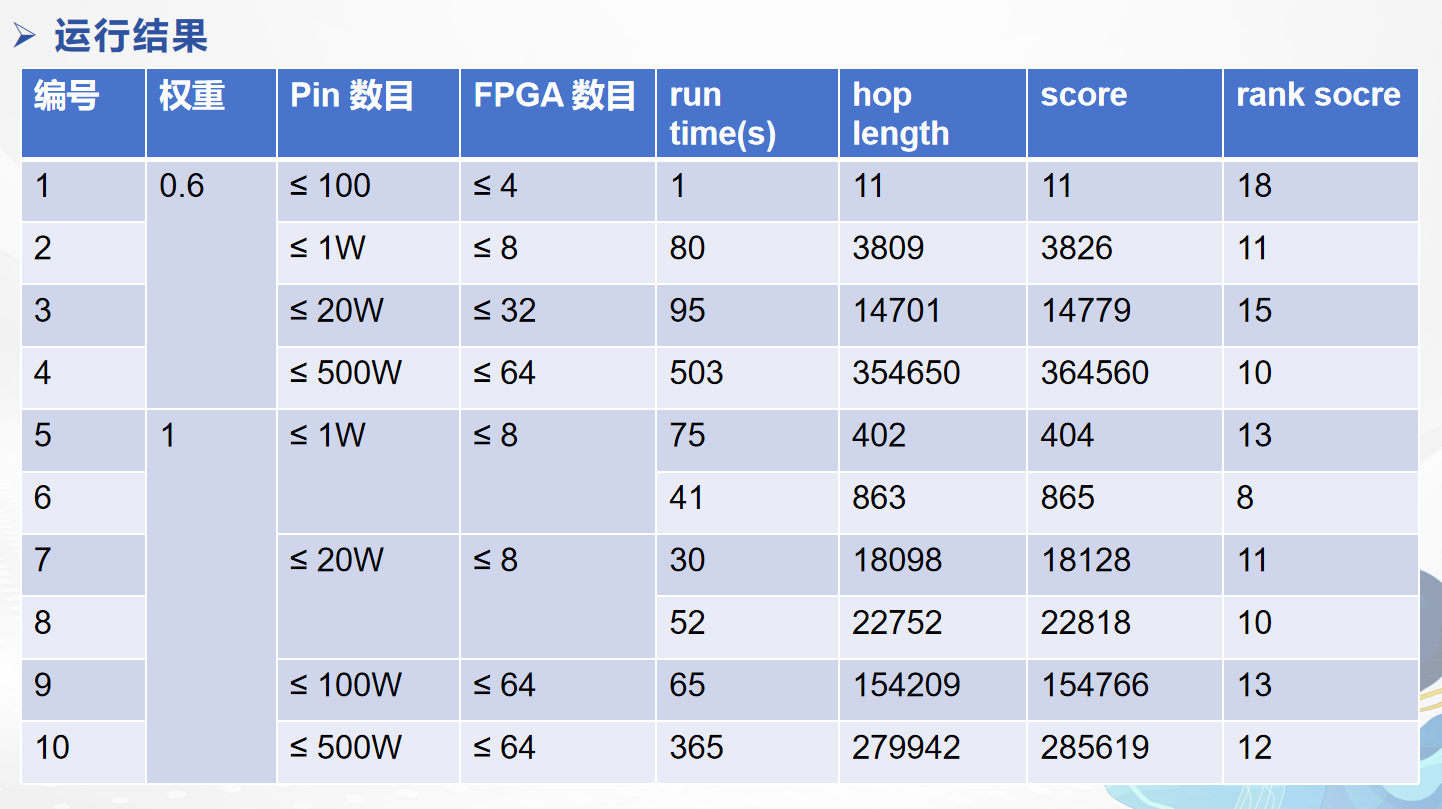
2按照FPGA的资源剩余量从大到小顺序；

3随机顺序。

最后选取hop和最优的结果输出。

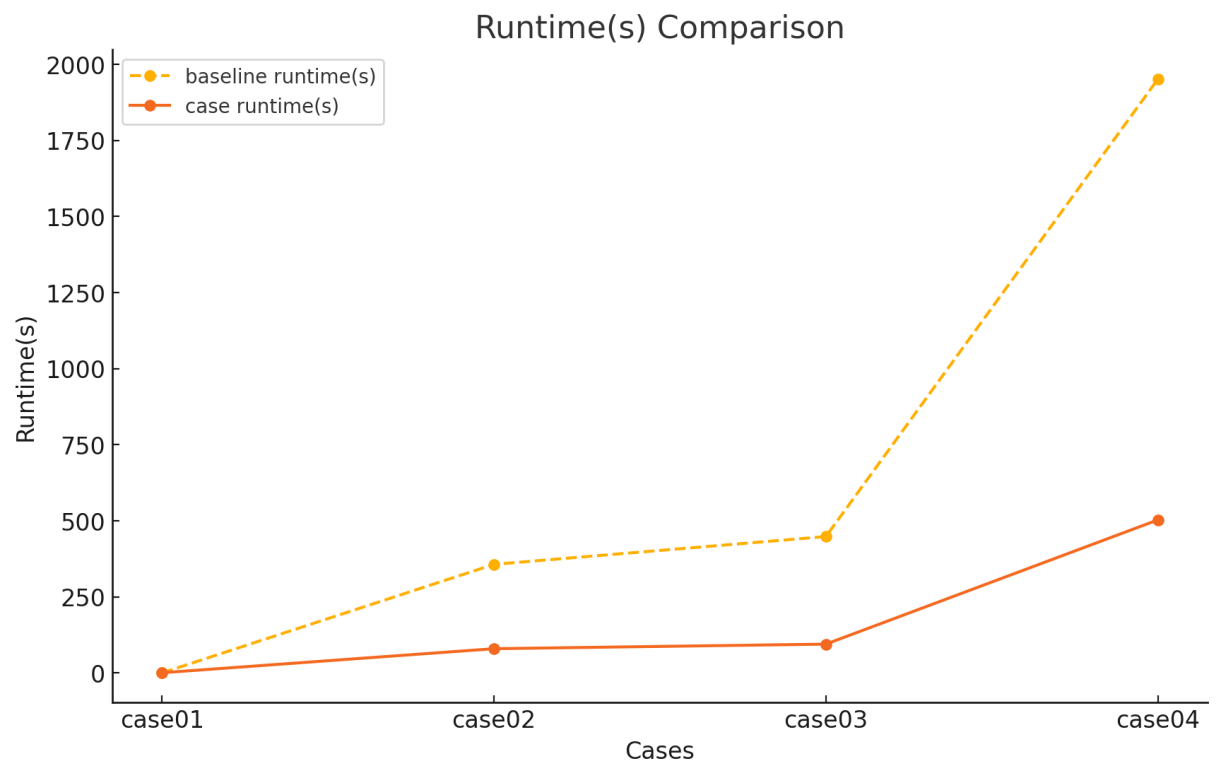
我们的算法在函数运行过程中多次释放不再使用的中间变量，压缩占用的内存空间，从而可以在内存限制下多线程并行程序。在主函数内读取了输入文件后，我们会估算各线程的内存占用，得到最大可用的线程数。同时使用智能指针管理内存，在内存超出限制时自动释放线程防止超出内存限制。

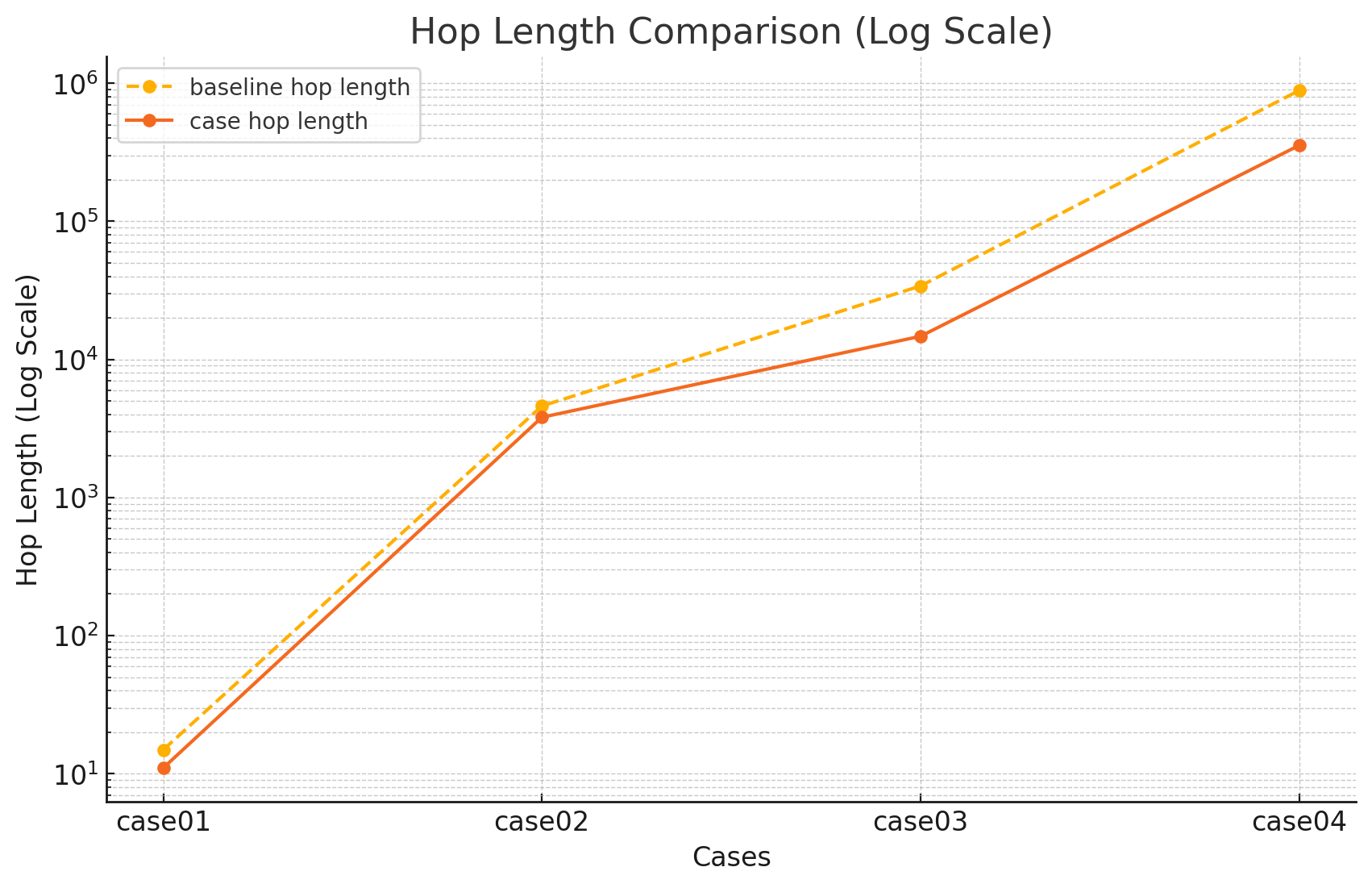
运行结果

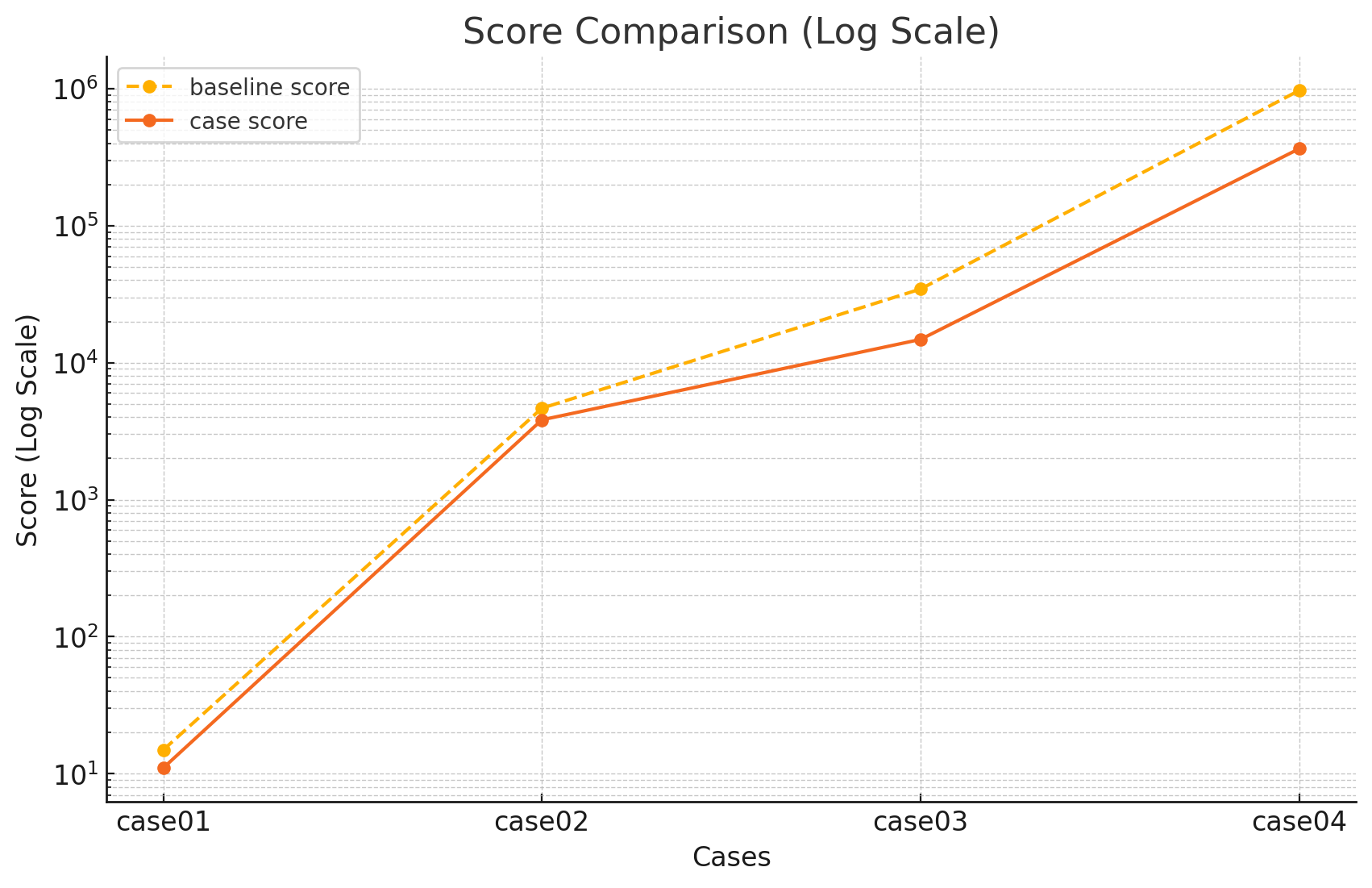


从上面的结果可以看到我们在每个case的运行时长均不超过10分钟，求解十分快速。

由于赛题方只放出了公开case预提交的具体结果，所以我们选取预提交中公开数据的平均值作为baseline。

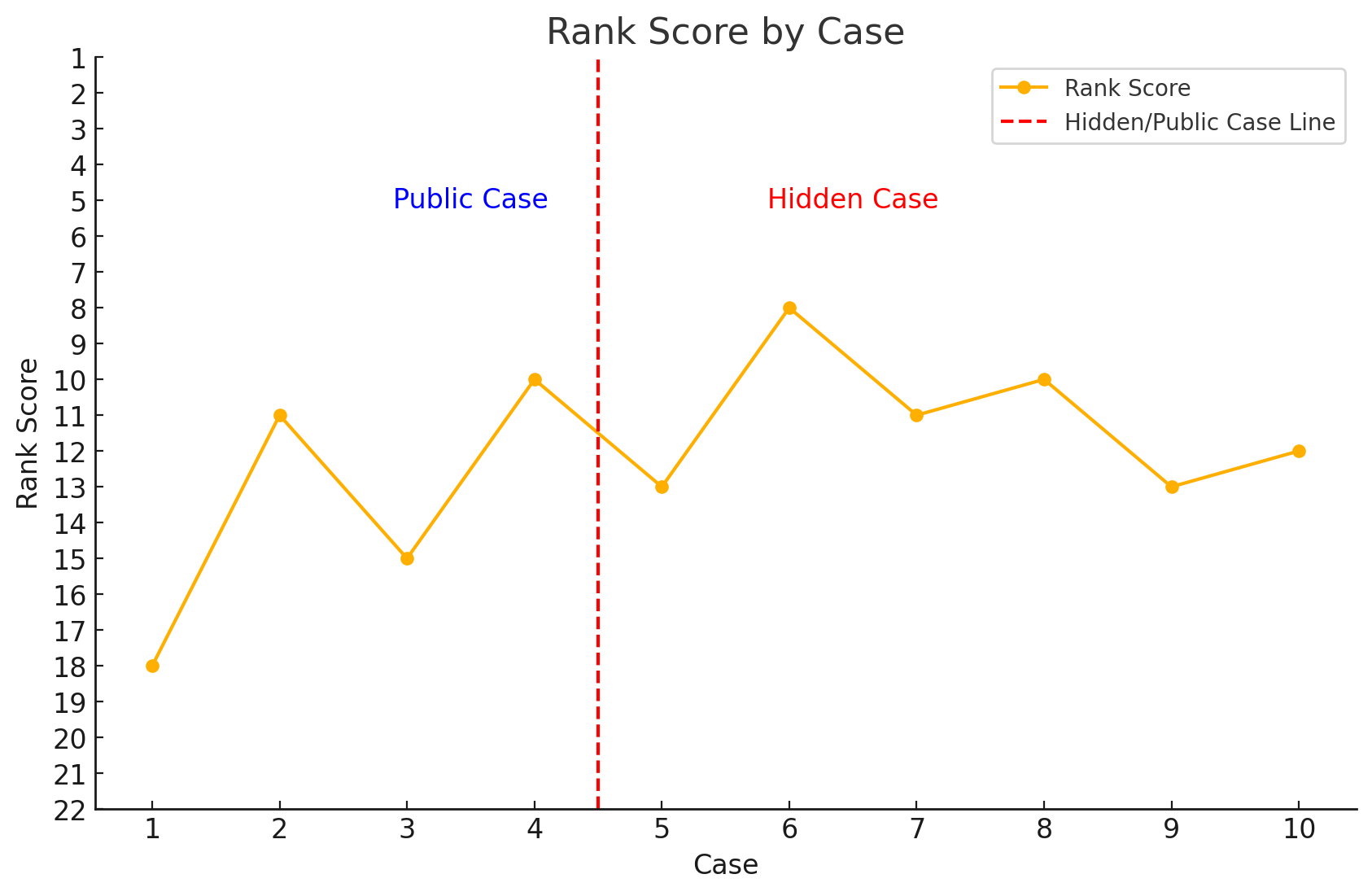




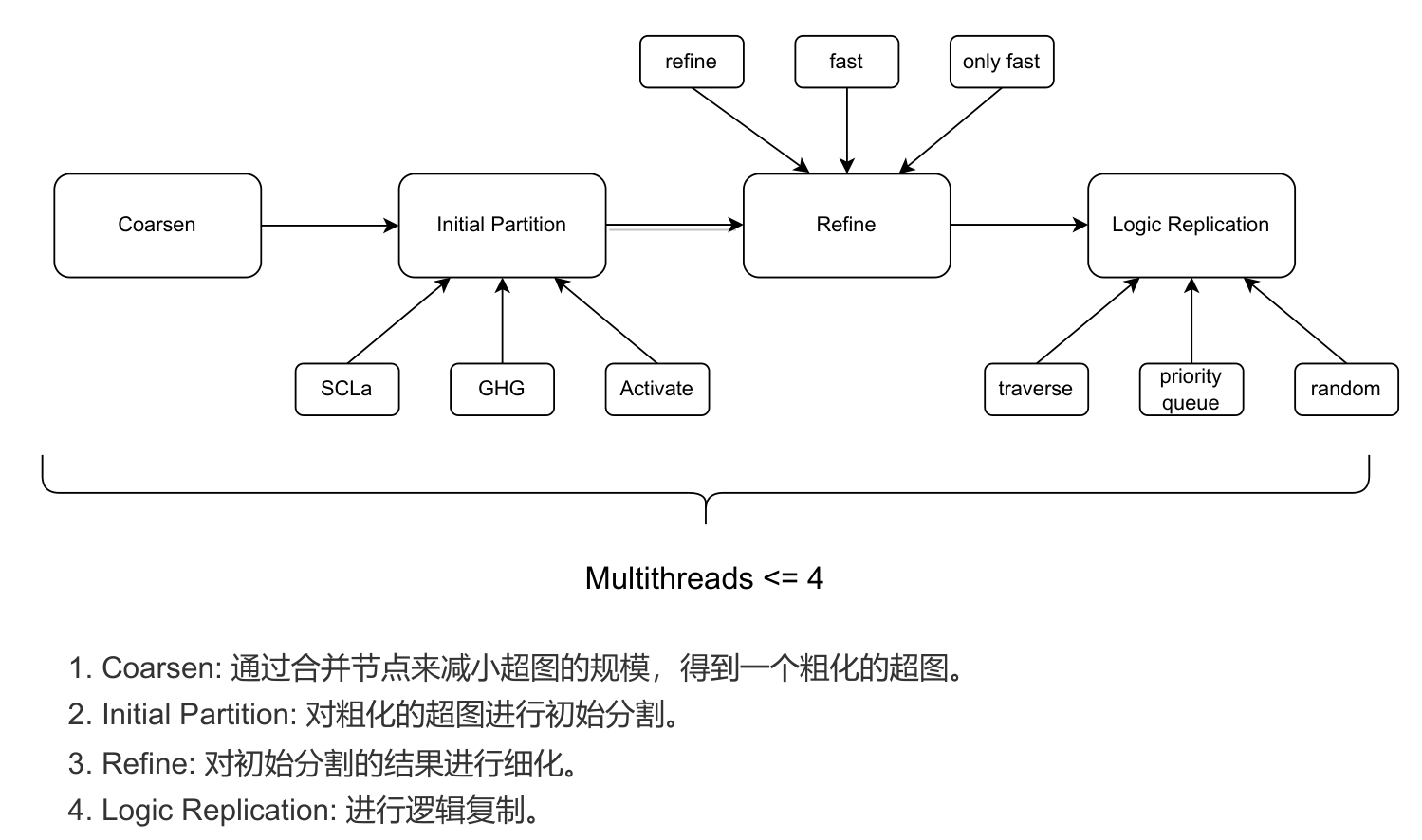


（score图加说明，分越低越好）

在上面的图中，橙色的为我们算法的结果，黄色为baseline结果，可以看到相较于baseline，我们的算法在样例较大的情况下在求解时间优于baseline，速度相较于baseline平均提升3.5倍，hop路径相较于baseline平均提升60%，最终在分数上相较于baseline平均提升59%。



从全部测试的分数排名中可以看到我们的算法在隐藏case中的表现比公开case要好，说明我们算法的泛化能力很强。



总结展望

创新点

我们通过多种评分函数优化贪心算法的分割过程，提升了接近全局最优解的能力。同时我们根据电路规模动态调整算法步骤，平衡计算时间与结果质量。我们还使用了智能指针管理内存，在内存接近溢出时自动释放资源，提高内存利用率。

展望优化方向与反思

因为我们程序的运行时间距离每个case 1小时的时间上限还有较大余裕，因此有充分的空间引入其他算法，我们可以考虑以下优化手段：

1. 探索其他加速方法：考虑通过优化算法实现或引入新的技术手段提升计算速度；
2. 改进逻辑复制算法：优化逻辑复制策略，提前为复制操作预留空间；
3. 引入遗传算法：利用历史结果优化搜索过程，避免陷入局部最优，寻找全局最优解；

反思

我们在数据结构的选用上使用了C++的标准库中的链表，但是我们也可以通过引入自定义的大根堆数据结构，通过 shift 操作在 O(log n) 时间复杂度内修复堆结构，从而避免每次操作时维护多个不准确且重复计算的小堆。这种方法可以直接在全局范围内操作堆，提升性能并减少不必要的计算开销。